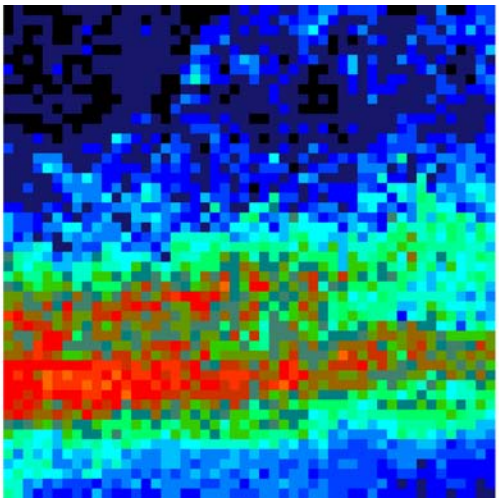
NOAA Technical Report NESDIS 139

Computing Applications for Satellite Temperature Datasets: A Performance Evaluation of Graphics Processing Units



Washington, D.C.
December 2011

U.S. DEPARTMENT OF COMMERCE
National Oceanic and Atmospheric Administration
National Environmental Satellite, Data, and Information Service

NOAA TECHNICAL REPORTS

National Environmental Satellite, Data, and Information Service

The National Environmental Satellite, Data, and Information Service (NESDIS) manages the Nation's civil Earth-observing satellite systems, as well as global national data bases for meteorology, oceanography, geophysics, and solar-terrestrial sciences. From these sources, it develops and disseminates environmental data and information products critical to the protection of life and property, national defense, the national economy, energy development and distribution, global food supplies, and the development of natural resources.

Publication in the NOAA Technical Report series does not preclude later publication in scientific journals in expanded or modified form. The NESDIS series of NOAA Technical Reports is a continuation of the former NESS and EDIS series of NOAA Technical Reports and the NESC and EDS series of Environmental Science Services Administration (ESSA) Technical Reports.

An electronic copy of this report may be obtained at
http://www.star.nesdis.noaa.gov/star/socd_pub.php

A limited number of copies of earlier reports are available by contacting Susan Devine, NOAA/NESDIS, E/RA, 5200 Auth Road, Room 701, Camp Springs, Maryland 20746, (301) 763-8127 x136. A partial listing of more recent reports appears below:

NESDIS 110 **An Algorithm for Correction of Navigation Errors in AMSU-A Data. Seiichiro Kigawa and Michael P. Weinreb, December 2002.**

NESDIS 111 **An Algorithm for Correction of Lunar Contamination in AMSU-A Data. Seiichiro Kigawa and Tsan Mo, December 2002.**

NESDIS 112 **Sampling Errors of the Global Mean Sea Level Derived from Topex/Poseidon Altimetry. Chang-Kou Tai and Carl Wagner, December 2002.**

NESDIS 113 **Proceedings of the International GODAR Review Meeting: Abstracts. Sponsors: Intergovernmental Oceanographic Commission, U.S. National Oceanic and Atmospheric Administration, and the European Community, May 2003.**

NESDIS 114 **Satellite Rainfall Estimation Over South America: Evaluation of Two Major Events. Daniel A. Vila, Roderick A. Scofield, Robert J. Kuligowski, and J. Clay Davenport, May 2003.**

NESDIS 115 **Imager and Sounder Radiance and Product Validations for the GOES-12 Science Test. Donald W. Hillger, Timothy J. Schmit, and Jamie M. Daniels, September 2003.**

NESDIS 116 **Microwave Humidity Sounder Calibration Algorithm. Tsan Mo and Kenneth Jarva, October 2004.**

NESDIS 117 **Building Profile Plankton Databases for Climate and EcoSystem Research. Sydney Levitus, Satoshi Sato, Catherine Maillard, Nick Mikhailov, Pat Cadwell, Harry Dooley, June 2005.**

NESDIS 118 **Simultaneous Nadir Overpasses for NOAA-6 to NOAA-17 Satellites from 1980 and 2003 for the Intersatellite Calibration of Radiometers. Changyong Cao, Pubu Ciren, August 2005.**

*Cover image compares a single 50 km pixel with equivalent 1 km pixels.

NOAA Technical Report NESDIS 139

Computing Applications for Satellite Temperature Datasets: A Performance Evaluation of Graphics Processing Units

Timothy F.R. Burgess
NOAA/NESDIS/STAR Coral Reef Watch
675 Ross River Road
Townsville, Queensland 4817
Australia

Scott F. Heron
NOAA/NESDIS/STAR Coral Reef Watch
675 Ross River Road
Townsville, Queensland 4817
Australia

Washington, DC
December 2011

U.S. DEPARTMENT OF COMMERCE
Gary Locke, Secretary

National Oceanic and Atmospheric Administration
Dr. Jane Lubchenco, Under Secretary of Commerce for Oceans and Atmosphere
and NOAA Administrator

National Environmental Satellite, Data, and Information Service
Mary Kicza, Assistant Administrator

Contents

Figures

Executive Summary

The Coral Reef Watch program of NOAA develops and provides remote sensing tools for the conservation of coral reef ecosystems. Reef managers and other stakeholders have expressed a desire for higher resolution monitoring tools than those currently available. In moving to higher resolution global products, Coral Reef Watch faces the challenge of *orders of magnitude increase* in the size of datasets. Traditionally this would mean simply upgrading to faster x86 Intel-based systems however Intel performance per processor has peaked due to frequency scaling issues. Newer Central Processing Unit (CPU) development has adopted a path of multiple cores rather than increasing an individual CPU's performance. To take advantage of newer hardware, a change to parallel programming methods is required and fortunately satellite datasets are relatively well suited to parallel algorithms.

However, a newer form of commodity parallel hardware, the Graphic Processing Unit (GPU) is significantly advancing in both speed and performance, pushed hard by the demand of consumer gaming enthusiasts. Companies such as AMD(ATI) and NVIDIA, recognizing the applicability of the latest hardware for high performance and scientific computing, have provided tools to allow programmers to develop software for these numeric computing engines. For tasks that lend themselves to parallel processing, this type of program, known as General Purpose GPU (GPGPU) computing is providing improved performance compared to solely using traditional CPUs.

We compare the performance of a current high-end quad core Intel CPU and a 480 core NVIDIA GPU with two representative algorithms on a well-known climate record quality dataset – the Pathfinder SST dataset. This dataset currently consists of 29 years of global daily SST measurements at 4 km resolution – approximately 33 million pixels for each day.

Two algorithms representing a very simple (SST persistence gap fill) and a more complex (SST exponential decay over time) gap filling algorithm were used to assist with the comparative assessment. Although comparing computational performance, I/O (input/output) performance is also important due to the large quantity of data and so we optimize I/O with advanced filesystems and a solid-state drive (SSD).

The GPU processing shows a marked improvement (2.4-times faster) over the CPU for the simple algorithm. For the more complex algorithm, the performance increase improves to a factor of 3.3. It is of note that the time required to input and output data is significant in the context of the gap filling algorithms employed.

The disadvantages of utilizing GPU for computing are the requirement of using a more rigorous programming language (e.g., 'C') rather than a high-level programming language such as IDL, which is widely used in the field of satellite remote sensing, and being familiar with the programming techniques and constraints related to the GPU software development tools. However the performance benefits are significant, demonstrating that using GPU tools can reduce computation time several-fold.

1. Introduction

Coral Reef Watch (CRW) is a program of the National Environmental Satellite Data and Information Service (NESDIS) of the U.S. National Oceanic and Atmospheric Administration (NOAA) and a part of the NOAA Coral Reef Conservation Program (CRCP). Coral Reef Watch's mission is to provide remote sensing tools for the conservation of coral reef ecosystems. Through these remote sensing tools, CRW assists in the management, study, and assessment of impacts of environmental change on coral reef ecosystems. Over more than a decade, Coral Reef Watch has developed a decision support system including many suites of monitoring and outlook products for coral reef managers and scientists. Coral Reef Watch's operational satellite data products, such as coral bleaching HotSpots and Degree Heating Week products based on satellite sea surface temperature (SST) measurements, provide current reef environmental conditions globally to quickly identify areas at risk for thermally-induced mass coral bleaching. Prolonged high thermal stress can cause corals to lose their symbiotic algae and thus effectively starve. In the event of severe thermal stress, disease and mortality may follow. At present, CRW also provides experimental products that monitor ocean acidification; duration of low-wind conditions; short-term trends in SST; bleaching potential based on combined effects of light and temperature; the risk of coral disease outbreak; and climate model-based seasonal outlooks of the potential for bleaching-level stress.

Mass coral bleaching events are well correlated with thermal stress (e.g., Dennis and Wicklund 1993, Winter *et al.* 1998, Hoegh-Guldberg 1999, Berkelmans 2002). Thus, continuous monitoring of SST and bleaching thermal stress at the global scale provides reef managers, researchers and stakeholders with critical information to understand, predict and monitor the development of mass coral bleaching and assist in improved, informed management and guiding *in situ* survey should mass coral bleaching occur.

Current technologies cannot provide gapless global monitoring of SST due to various atmospheric and oceanic issues, including the presence of cloud, high variability of coastal-zone water temperatures, and satellite coverage. Thus, the provision of a gap-free SST dataset is of critical importance to the thermal stress monitoring by CRW, and in turn, resulting predictions of coral bleaching incidence and severity.

To date, CRW's satellite products have been produced at one half-degree spatial resolution; that is, roughly 50 km in pixel size; since the 1990s. During the past few years, global satellite SST datasets at higher spatial resolutions, including 11, 7, 4, and 1 km, have become available; through the same period, coral reef stakeholders have requested finer-scale monitoring tools to aid in management. As a result, CRW has been developing next generation products utilizing the higher resolution global near real-time satellite data for improved monitoring and prediction of the coral reef environment. However, there are significant scientific and computational challenges in migrating to higher resolution on both regional and global bases and providing proven, trustworthy products for coral focused decision-making. The computational challenges and proposed solution will be discussed in this report.

Analysis and processing of global resolutions at 50 km, while demanding, is not overwhelming for current x86 desktop systems. However the source data, and thus CRW's development and production of next generation products, are moving to higher resolutions such as 4 km and 1 km. A resolution change from 50 km to 4 km alone means *over two orders of magnitude increase in*

data size. Regional datasets continue to be feasible in short timeframes; however, processing of a global dataset at 4 km and higher resolution becomes very lengthy using CRW's traditional data processing methods (i.e., the use of standard x86 Central Processing Unit (CPU) machines). Not only will the length of a single processing run significantly increase but also the time required for scientific analysis and the '*change->validate->change->validate'* software development cycle will increase by even higher magnitudes. The increase in processing times can be of the order of days rather than hours.

The method traditionally used to deal with increased data volume – simply acquiring 'faster' computers – is unfortunately not as applicable as it used to be. The approach known as frequency scaling, whereby semiconductor manufacturers increase CPU frequency, has driven faster and faster CPUs over time. However, frequency scaling effectively ended in 2004 when Intel cancelled their single core development program[1]. As further frequency scaling causes overheating in the CPU unit, Intel moved to multi-core development that builds multiple cores into a single integrated circuit die. With parallel computing now the dominant paradigm in computer architecture, taking advantage of parallel computing is of significant interest in the software development community.

However, the amount of performance gained by the use of a multi-core processor depends very much on the software algorithms and implementation. In particular, the possible gains are limited by the fraction of the software that can be parallelized to run on multiple cores simultaneously. Thus, to enjoy performance improvements with each new generation of microprocessors will require software consisting of parallelized programs, in which multiple threads of execution cooperate to achieve the result faster.

Multi-core CPU systems certainly extend the accessibility of parallel hardware and boost the computing speed significantly, but at the same time, the technical advance in a relatively new class of device widely available on most of computers, the 'many-core' Graphics Processing Unit (GPU), allows for boosting the computing speed to an even greater extent.

The popularity of consumer computer games has significantly accelerated the adoption of 3D graphics in consumer computing. Beginning with transform and lighting computation, there has been a progression to the point now where the software developer has control over the exact computations that can be performed. Recognizing that markets exist beyond game graphics, GPU vendors have included new components designed strictly for general-purpose computation. These general-purpose GPU (GPGPU) programs address problems in non-graphics domains (Owens *et al*. 2007).

Coral Reef Watch conducted an investigation to evaluate the performance differential between a recent high-end 8-core Intel CPU and a 480-core NVIDIA GPU for two representative satellite SST data processing tasks. The Version 5 NOAA/NASA Pathfinder 4 km SST dataset was used in the test because of its high quality, high spatial resolution, and long time series for a satellite

[1] New York Times (May 8, 2004) – Intel halts development of 2 new microprocessors, http://www.nytimes.com/2004/05/08/business/08chip.html?ex=1399348800&en=98cc44ca97b1a562&ei=5007 (accessed September 10, 2010).

data record. Additionally, this dataset will be used as a primary historical dataset for developing CRW's next generation products.

The design of this investigation and its results, along with the background on the GPU computing and the Pathfinder dataset, are described and discussed in this report.

2. High Resolution Satellite Data

The Advanced Very High Resolution Radiometers (AVHRR) onboard NOAA's Polar-Orbiting Environmental Satellites (POES) enjoy an unbroken, nearly 30-year history of observation from the same class of instrument (Kilpatrick, *et al.* 2001). These AVHRR Sea Surface Temperature (SST) observations date from 1981 and will continue perhaps another five to ten years into the future onboard the remaining NOAA polar-orbiting platforms and the European MetOp satellites. However, changes of data source as satellites are replaced, changes in instrument sensitivity during their lifetime, and changes in satellite orbit have resulted in variations in the precision, accuracy, and mean of the AVHRR SST data through time. The Pathfinder SST program has a long history of reprocessing AVHRR observations to create accurate and consistent SST data records. Over more than a decade, Pathfinder SST has evolved over many versions with improved consistency and increased spatial resolution. At Version 5.0, it provides a global daily (daytime and nighttime) SST dataset at 4 km resolution for 1985-2009, with a new supplemental Version 5.1 that extends the time series to include the data for 1981-1984.

NOAA Coral Reef Watch's (CRW) operational product suite began in December 2000; as such, CRW has employed the Pathfinder datasets as a core long-term SST climate record dataset for coral reef environmental assessment and its satellite product improvement and development. At 4 km spatial resolution, a single piece of Pathfinder Version 5 daily daytime or nighttime global coverage data encompasses 8192 by 4096 pixels, which gives roughly 33 *million* pixels per day, for almost every day in a near 30-year period from 1981 to 2009. This dataset is expected to add new years on an approximately annual basis.

The SST values, quality assessment, and associated metadata are stored in HDF4 (Hierarchical Data Format 4) – a commonly used scientific data format. A quality flag, which ranges from 0 to 7 with 0 unusable and 7 at highest quality, is assigned to each SST value, indicating the overall quality of each SST value based on a hierarchical suite of quality screenings performed. A low quality assessment can result from any combination of cloud cover, sun glint, large sensor scan angle, or inconsistency with climatological reference temperature and/or with SST observations at neighboring pixels, among others. To reduce the size of the dataset, SST values are stored as 16-bit integer format ('short integer') that preserves the precision of the original floating type (satellite SST values are accurate to the first decimal place) by applying a scale factor and an offset. Quality flag data are simply in 8-bit format.

The size of this dataset provides challenges for timely data processing and analysis. Using CPU only computation, input/output (I/O) subsystem performance bounds can be reached as both SST and quality flag data at 33 million data elements per day are read and a corresponding per pixel output produced. Computational constraints also can be reached when using algorithms that perform temporal and/or spatial calculations. An improved computational mechanism was both desired and necessary.

The next Pathfinder SST dataset, Version 6, is expected to include daily data at 1 km spatial resolution, incorporating high-resolution passes of significant coral reef areas, namely the Caribbean, Eastern Australia and parts of the Western Pacific (Casey, *et al*. 2010). Thus, *an additional order of magnitude increase from 4 km* will occur in the not too distant future. This emphasizes the importance of having appropriate computational mechanisms in place to ensure reasonable time for product development and timely processing of near real-time decision support system products, for both providing reef managers and stakeholders critical information with resolution appropriate to their needs and exploring scientific opportunities that much higher resolutions may offer.

In addition to needing to process Pathfinder datasets, the next generation Coral Reef Watch decision support system will provide higher-resolution products, fully utilizing the high resolution near real-time global satellite data that have recently become available at 11, 5, and even 1 km spatial resolutions. The new generation products will be produced on a daily basis. Hence, high computing performance is required to facilitate such a decision support system for coral reef management.

3. IDL Programming Language

IDL (Interactive Data Language) is a programming language developed, provided and licensed by ITT Visual Information Solutions and is commonly used for interactive processing, analysis, and visualization of large amounts of data, including image processing. It has been widely used in the field of environmental satellite remote sensing and is one of main scientific computer programming languages used intensively in NOAA NESDIS, including the Coral Reef Watch program. The language does not require the user to specify types (it is dynamically-typed) or manage memory allocation/de-allocation (it uses automatic garbage collection) and only requires partial compilation to an intermediate code. It is vectorized; i.e., designed to allow the same operation (add, sum, mean, etc.) to occur over a single vector, matrix or higher dimensional arrays that are common in scientific datasets. The compiler and debugger are packaged into the open source Eclipse IDE and provided as a single product.

The IDL programming environment has the benefits of being an ‘all-in-one’ piece, making it simple to install. The dynamic typing relaxes syntax constraints for the programmer (however, errors are more likely caught in runtime than at compile time), and the array processing capability and interpreted nature make it suitable for both scientific dataset analysis and non-specialist programmers such as scientists.

For multi-core CPU work, the disadvantage with IDL is that thread support (i.e., the ability to run multiple processes) is not available to the programmer[2]. Instead, the programmer calls language functions, which may be threaded internally. Dependent on the internal threading (which is unknown to the programmer), multiple cores may or may not be used.

[2] Why not expose threads at the IDL user level? http://www.ittvis.com/services/techtip.asp?ttid=3252#Q12 (accessed September 10, 2010)

With the continuous improvement, enhancement, and evolution of IDL, currently at Version 8.0, it has become more sophisticated and now includes functionality that is available in other widely used scientific programming languages. As a result, NOAA NESDIS will likely continue to use IDL for the foreseeable future and IDL will remain an important programming tool to Coral Reef Watch. Hence, in this report, IDL programming will be investigated to evaluate its performance in some selected intensive tests of data manipulation and processing of high resolution Pathfinder SST on a high-end multi-core CPU computer and to compare with GPU programming.

4. General Purpose GPU Computing Architecture

The Central Processing Unit (CPU) is the primary element of a computer system carrying out the computer's fundamental functions and the instructions of computer programs, including the basic arithmetical, logical, and input/output operations of the system. The term CPU has been used since at least the early 1960s. The design and implementation of CPUs have evolved dramatically over time, but not until mid-2000s were multi-core CPUs developed and implemented on most computers.

A Graphics Processing Unit (GPU) is a specialized microprocessor designed to offload graphics rendering from the main processor to accelerate computing. It is now commonly used in computers, mobile phones, and game consoles. Starting to take form in the 1970s, GPUs were not programmable by the user until the 2000s. Present mainly on computer's video cards and occasionally on motherboards, modern GPUs are very efficient at rendering and manipulating computer graphics through their highly parallel structure, which makes them more effective and much faster than general-purpose CPUs for a range of complex algorithms. Nowadays, more than 90% of desktop and notebook computers have integrated GPUs. A modern GPU can contain hundreds of cores that allow parallel processing for hundreds of threads.

A GPU does not take away the fundamental functions carried out by a computer's CPU and does not control the instructions of computer programs, but it can share and carry out many computationally-intensive tasks that were previously carried out only by the CPU – running much more efficiently and faster by threading computing tasks to hundreds of GPU cores. This is not done internally by a computer system, but through user GPU programming. Only certain processes can be performed on GPUs and these programs almost always use the dedicated GPU memory.

A CPU seeks to run one (or a few) processes (or more strictly speaking 'threads') as fast as possible, depending on the number of cores that it contains (usually less than 10). A GPU seeks to run a very large number of threads concurrently, albeit perhaps individually more slowly than a CPU. The latter objective arises from the primary historical function of the GPU which was (and often still is) the offloading of enormous numbers of floating point calculations for 2D or 3D graphics purposes. This GPU design philosophy is forced by the fast growing video game industry that exerts tremendous economic pressure for the ability to perform a massive number of floating-point calculations per video frame in advanced games. Thus, GPUs are essentially numeric computing engines and have been designed from the outset for parallel processing.

GPUs will continue to evolve at a fast pace. Both Intel and AMD, two major chip manufacturing companies, have publically stated they will be incorporating GPUs onto the same die as the CPU

in future planned architectures[3,4] and have released initial products. With the future ubiquitous presence of GPUs, software and software development tools have been developed and will evolve to take advantage of the powerful GPU capability. The applicability of GPUs to high performance computing has become more and more evident, resulting in newer generations of GPU architecture and associated support software for general-purpose GPU (GPGPU) programs. NVIDIA's recent GeForce 400 series, often referred to as the 'Fermi' architecture, is an example of such, employing *3 billion* transistors. The 400 series card supports (in some options):

- C and C++ programming languages
- IEEE-754-2008 compliant 32-bit and 64-bit precision
- Up to 512 cores
- Error Correction Code (ECC) memory
- Concurrent execution of different kernel functions
- Up to 6GB of high-speed GDDR5 Dynamic Random Access Memory (DRAM)

Figure 1. NVIDIA GeForce 400 series card – GTX480 (http://en.wikipedia.org/wiki/File:Nvidia_GeForce_GTX480.jpg, 2010).

NVIDIA uses the branding 'Tesla' to refer to GPU hardware specifically designed for high-performance computing. Australia's Commonwealth Scientific and Industrial Research Organisation (CSIRO) has invested in Tesla GPU hardware to build a supercomputer cluster[5] and more GPU cluster developments are likely to occur in the near future. 'Fermi' class Tesla hardware became available to the general public in late 2010.

[3] The AMD Fusion Family, http://sites.amd.com/us/fusion/APU/Pages/fusion.aspx (accessed September 10, 2010)

[4] CNET, (September 16, 2010) – Intel's Sandy Bridge graphics tech: How good is it? http://news.cnet.com/8301-13924_3-20016628-64.html (accessed September 20, 2010)

[5] Speeding Up Science, CSIRO's CPU-GPU Supercomputer Cluster, http://www.csiro.au/resources/GPU-cluster.html (accessed September 10, 2010)

A low cost (acquired at $USD360) Fermi class GPU with 480 cores and 1GB of DRAM was chosen for our test and commands compiled in the C language were used to run the test.

4.1 C Programming with CUDA extensions

The C programming language, developed in 1972 and strongly associated with UNIX, is one of the most popular programming languages of all time. In November 2011, it retained the second position on a widely regarded ranking list of popular programming languages[6].

As a 'low-level', rather than a higher-level (like IDL), language, the C language embodies minimalism and can produce very high performance programs. However, its low-level nature, that requires intensive programming to communicate directly with computer's operating system, including memory management, makes it less suitable for non-specialist programmers than other higher-level languages. Comparatively, IDL, as a high-level language, is easier for non-specialist programmers to develop software.

CUDA (an acronym for Compute Unified Device Architecture) is an extension to the C language developed by NVIDIA to allow software developers access to the parallel computation elements in the CUDA-capable GPUs. Recent versions of the CUDA toolkit also support the use of C++.

The CUDA extensions allow a programmer to define a C function to be run on each of the hundreds of GPU cores. This function is generally referred to as a 'kernel' to identify it from other functions that run on the CPU. A CUDA program typically loads data from the 'host' computer main memory to high-speed memory on the GPU, runs the kernel in a highly parallel form across all the GPU cores and copies the resulting output from the GPU memory back to the host memory. Defining a 'kernel' function and executing it across a large array of data is known as a Single Instruction-Multiple Data (SIMD) approach. As a general rule, employing a kernel by the SIMD approach tends to be conceptually simpler to its programmer than managing multiple CPU threads.

CUDA toolkits are available for Windows, Linux, and Mac OS X platforms at no cost. These toolkits provide GPU drivers, a compiler, and a debugger, for writing and debugging GPU code.

4.2 Software tools for IDL to utilize GPU computing

Some software tools have been developed to facilitate the use of high performance computing resources available on modern GPUs by non-specialist programmers who exclusively develop software using high-level programming languages. Such programmers include engineers, scientists, analysts, and other technical professionals.

GPULib is a product that consists of a library of mathematical functions developed by Tech-X Corporation (http://gpulib.txcorp.com) for high performance data analysis using GPUs. The

[6] TIOBE Programming Community Index. http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html (accessed November 20, 2011)

library is built on top of NVIDIA's CUDA platform (described in the previous section). GPULib targets engineers and scientists who prefer to focus in their domains of expertise, but must do some amount of software development to get their job done using high-level programming languages. GPULib allows these users to access high performance computing with minimal modification to their existing programs. GPULib can accelerate new applications or be incorporated into existing applications with minimal effort because it provides programming interfaces for a number of Very High-Level Languages (VHLLs), including IDL. No knowledge of GPU programming or memory management is required. This software tool would likely benefit the development in future applications of GPU technology for CRW by making GPU programming more accessible to scientific programmers. However, the added expense combined with an expectation that the GPU performance using this tool would likely lie somewhere between that of native IDL and native C with CUDA resulted in a choice to not incorporate this product in the test described here.

5. Performance Tests

5.1 Test Algorithms

Missing values in satellite data are a common problem in remote sensing that can be resolved using temporal and/or spatial gap filling techniques. Data gap filling is performed for locations and times for which the measured SST values are deemed unusable, as the result of cloud cover, sun glint, large sensor scan angles, etc. This is the most time consuming and computationally intensive process among all the computation needs in CRW product development and routine product generation. Hence, data gap filling has been chosen for evaluating computational performance using GPUs.

For the purpose of testing the platform's computational abilities, data gap filling was carried out using two separate algorithms. The first algorithm is a simple persistence model. At pixels where the Pathfinder data are deemed to be unusable, the algorithm persists the prior high quality retrieval for that pixel until a newer high quality retrieval is acquired; i.e.,

$$T_s(t+1) = T_s(t), \qquad (1)$$

where T_s is the SST and t represents time. A benefit of this approach is that all data are temperatures that have physically occurred at the location, as opposed to deriving values from existing high quality values from different times and/or other locations (e.g., by interpolation). It is of note that persistence is currently employed in the NOAA Coral Reef Watch near real-time global 50 km SST product (Skirving et al. 2006) and also the ReefTemp product (Maynard et al. 2008), which monitors temperature-related bleaching risk for the Great Barrier Reef region of Australia.

The second algorithm is designed to specifically address one type of data gap: gaps due to cloud cover. Where cloud is present, the retrieval value can reflect, in full or in part, the temperature of the cloud top, rather than the temperature of the water below. Typically, when clouds persist over a patch of water during the daytime, they block the heating from the sun and the temperature of the sea-surface decreases. The rate and magnitude of cooling are usually dependent upon the factors of air temperature, humidity, and wind. Here we assume that the rate of cooling is

dependent only upon and proportional to the difference between the water and air temperatures and that the air temperature remains constant during the period of cloud cover:

$$\frac{dT_s}{dt} = -\frac{1}{\tau}(T_s - T_a), \qquad (2)$$

where T_s is the SST, t is time, τ is the proportionality constant and T_a is the air temperature. Integrating equation (2) yields:

$$T_s = T_a + \Delta T * e^{-t/\tau} \qquad (3)$$

and results in the exponential decay of SST toward the air temperature, where τ is revealed as the time constant of the decay and $\Delta T = T_s - T_a$ is the initial difference between air and water temperatures. In reality, not all low-quality values are due to clouds; however, this second algorithm simply calculates temperature values according to this equation to evaluate computational performance.

For the purpose of this exercise (i.e., to compare the computational performance of two hardware architectures), a constant value is prescribed for the water-air temperature difference along with a prescribed decay time constant. These were obtained by observing the temperature reduction through cloudy days at the Automated Weather Station of the Australian Institute of Marine Science located at Agincourt Reef in January 2009 (http://data.aims.gov.au/aimsrtds/faces/latestreadings.xhtml). A brief analysis of the data provided the values:

$$\Delta T = 0.75\,^{\circ}\mathrm{C} \qquad (4)$$

and

$$\tau = 0.887 \text{ days.} \qquad (5)$$

Using the prescribed value of ΔT, T_a at any pixel can be simply calculated by applying $T_a = T_s - \Delta T$, where T_s is the most-recently observed temperature at the pixel immediately prior to the data gap ($t = 0$ day).

Clearly, during the actual product development and scientific data analysis, the cause of a data gap would be determined first (e.g., using the quality flag) and then a more complex and robust algorithm would be used to determine SST to fill the gap. While the methodology presented here is overly simplistic, it is, from a computational test perspective, a good basic-level comparison to see the performance difference in the two approaches.

5.2 Test Computing Platform

The system used to perform the comparison was a 64-bit Linux (Ubuntu 10.04) platform running a custom-built kernel (version 2.6.35) with symmetric multiprocessing enabled. The CPU used was an Intel i7 950 at 3.07GHz with 24GB of system RAM. The 24GB RAM was fully addressable by a single 64-bit process but memory used never exceeded available memory.

It was regarded as important to relieve I/O bottlenecks in retrieving and writing data. To deliver sustained, high sequential read performance, the Pathfinder HDF4 input data were placed on a Crucial 300GB solid state drive (SSD), whose performance exceeded that of the SATA drive that houses the CRW data repository. This SSD was formatted, loaded with the Pathfinder dataset then symbolically linked into the data repository directory structure.
For output of sustained sequential writes, we opted for simplicity and capacity, and used a 1TB Western Digital SATA drive. While SSDs commonly exhibit excellent read performance, the same cannot be said for write performance and thus we elected to use a drive that exhibited good, consistent write performance. We reduced by half the output requirements by transposing each SST pixel value from a floating point of actual SST value into a 16-bit integer, using the same conversion as used in storing the Pathfinder source data: SST values, which are all greater than -3.0 °C, had 3.0 added to obtain all positive values and then multiplied by a factor of 100.0 to preserve two decimal places of accuracy. The remaining decimal places were then truncated and the values were saved in integer form.

Given the large amounts of data to be transferred, filesystem selection and tuning was considered important and investigated. Specifically, comparisons were made between the "ext2" (non-journaling) and the "ext4" (journaling, and other features) filesystems. A B-tree file system (Btrfs) was considered; however, the performance for the kernel version used seemed to not generally be on par with more predominant filesystems[7]. In file read/write testing, ext4 with journaling and barriers turned off provided consistently better performance than ext2 (which performs no file journaling). It is of note that ext4 supports 'extents,' a key benefit for large files that dramatically reduces system I/O calls, and this is likely the reason for the superior performance.

Journaling is a feature of modern filesystems where changes are logged to prevent corruption in the event of a crash or power loss. However, given performance comparison was the aim of this investigation and that the system in use ran on an uninterruptible power supply, we elected to disable journaling to take advantage of an approximately 3% enhancement in performance.

To ensure SSD performance did not degrade, and therein affect the results, *trim* command support (for flagging deleted SSD blocks) was enabled in the custom Linux kernel (version 2.6.35).

5.3 GPU programming

CUDA refers to system memory as *host* memory and memory on the GPU as *device* memory. In GPU programs, almost always, data is read from disk to *host* (system) memory and then copied to *device* memory where the GPU performs manipulation of the data. Latest editions of CUDA-compatible hardware can directly access *host* memory however NVIDIA strongly recommends manipulating data in *device* memory as performance is significantly faster. Thus, CUDA code

[7] EXT4 and Btrfs Regressions in Linux 2.6.36. http://www.phoronix.com/scan.php?page=article&item=linux_2636_btrfs&num=1 (accessed September 20, 2010).

typically copies data from host to device, manipulates said data and then copies data back to the host (usually for output). Thus, it is important to ascertain whether the GPU memory can accommodate an algorithm's requirements.

To begin with, we wrote simple C functions to read each piece of input data off the hard disk and place it into *host* memory. Using an array size encompassing the dimensions of the data, and given a date to read, these functions located the pertinent files, and then read a daily SST, the associated quality metadata and the landmask into host memory, respectively. The end result, we referred to as the *target* array and so wrote a function to take the target array from *host* memory and write to a disk file.

The final program contained preprocess, process, and endprocess functions:

- The preprocess function was called to allocate the required host and device (GPU) memory for SST, quality and landmask data; and a target array, equivalent to SST, to initially hold the previous day's gap-filled output, which is updated in both algorithms to become the current day's gap-filled output. For the more-complex test algorithm, device memory was also allocated to count the number of days since the most-recent high-quality value. The preprocess function also loaded the landmask data from disk into both host and device memory.
- The main program then ran a loop to consider each date of the Pathfinder dataset duration. Within the loop, the particular date's SST and quality flag data were loaded into host memory and the process function was then called. This function copied SST and quality flag data into device memory; performed the test algorithm, updating the target array elements in device memory with current day acquisitions or gap-filled values; and lastly, copied the target array into host memory. The final step of the loop was to write the target array from host memory to a disk file.
- The endprocess function released the allocated memory on the GPU device.

A separate process function was written for each algorithm with the pertinent process function linked in at compile time. Thus all code was the same for the two GPU programs, except for the process function.

The most fundamental aspect was that CUDA allows definition of a *kernel* function. Using CUDA syntax, this kernel is called a specific number of times *in parallel*. As we had 33,554,432 pixels, once all data were in device memory, we called the kernel function 33,554,432 times. CUDA supplies information to each kernel when run, essentially its index number. Our kernel function simply took its index number, used that index to extract the pertinent pixel value from each array and then wrote the calculated result into the target array index. As the gap filling algorithms were purely temporal (i.e., no spatial component), this made the kernel code very simple. CUDA makes no guarantees regarding the order or timing in which the kernel is called. This was not a concern to us as the algorithms worked on a per-pixel basis, which is compatible given the spatially independent nature of both algorithms.

5.4 Test Results

The results were obtained for a complete processing of the entire 1981-2009 Pathfinder dataset and included both the file-read and processing time (file-output is not included). Each test was

run twice and the average taken. Figure 2 summarizes the results for each algorithm and each platform.

For the simplest test (Algorithm 1 – Persistence gap fill), the GPU processing took 164 minutes, significantly faster than the CPU test (276 minutes). We believe this test became mostly I/O bound and the GPU runtime is almost completely a function of significant I/O requirements. To examine this, we profiled the Pathfinder HDF4 reads and determined that approximately 84 minutes (as reflected by the shaded area in Figure 2) was required to load the dataset into memory. Discarding this component of the run-time showed the GPU processing (80 minutes) was 2.4-times faster than the CPU processing (192 minutes).

For the second test (Algorithm 2 – Exponential decay gap fill), the difference was stark: the GPU completed in 187 minutes, whereas the CPU required 426 minutes. Excluding the portion of time required to read the dataset (84 minutes), the GPU (103 minutes) was 3.3-times faster than the CPU (342 minutes).

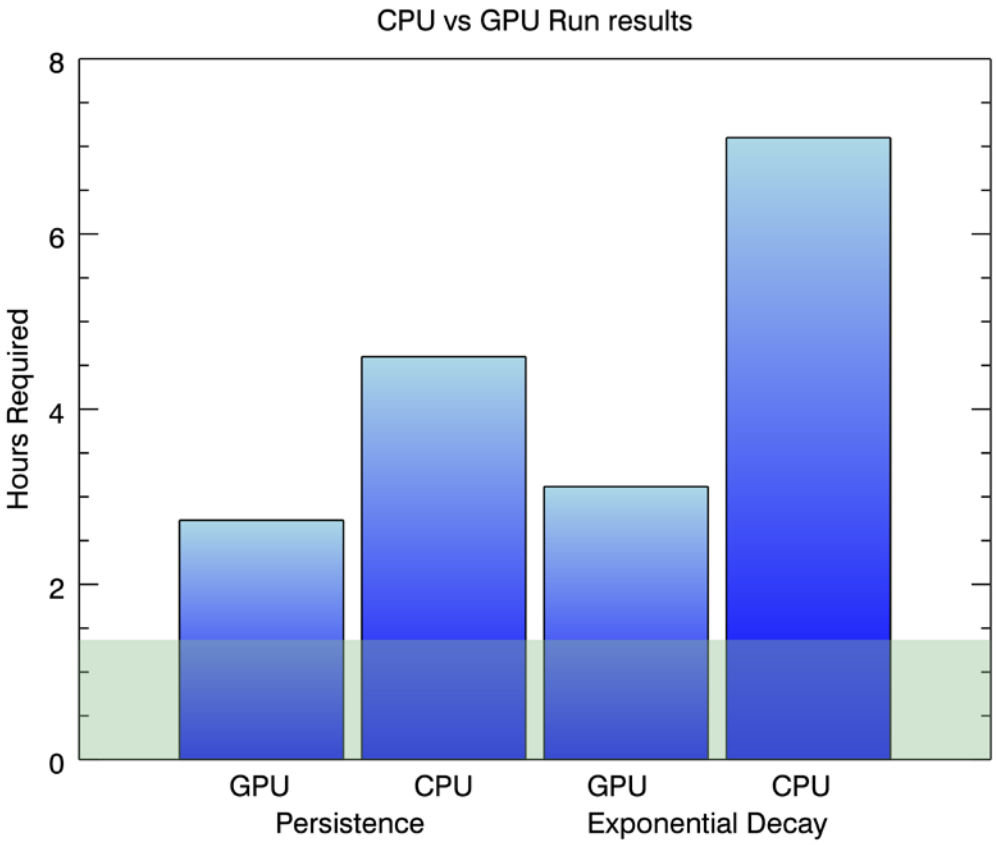


Figure 2. Timed runs of GPU and CPU on two test algorithms (simple persistence and exponential decay). The lower shaded area reflects the duration required for extracting Pathfinder HDF4 files from disk to host memory, which was measured as a separate serial task. Data writing was not easily measurable due to I/O subsystem buffering.

6. Discussions and Conclusions

The GPU method demonstrated a significant performance advantage over the CPU method. This was not unexpected but the difference is interesting. Although the exponential decay algorithm was only one order of complexity greater than the persistence algorithm, the GPU demonstrated a significantly increased performance advantage over the CPU. We surmise that as more complex algorithms (and larger datasets; e.g., 1 km resolution) are used, a GPU approach is likely to show even greater improvements.

We believe that as the data are compressed, the HDF4 library uses a single processor thread to uncompress data and this possibly contributes to a significant element of the run times. A different mechanism to load data may well reduce total run times substantially, however the focus of this investigation was a computational comparison and these detailed I/O optimizations are a consideration for future work. We simply state that both processor performance and I/O subsystem performance have important impacts on the complete duration required.

However it is important to note that a GPU approach is not a 'silver bullet' and that various constraints exist:

- GPU memory is currently limited to 6GB per card (our GPU model had 1GB total memory). While this was not an issue for the algorithms presented here, it may be a constraint for other memory-intensive algorithms.
- As all calculations occur on the GPU processors and in GPU memory, host system libraries cannot be used. GPU toolkits provide math libraries for common calculations, however any custom function library would need to be ported for GPU use. This does not impact the needs of Coral Reef Watch.
- As algorithms scale in complexity, the register requirements per processor increase. As this resource is shared across processors, less threads can be executed in parallel. However, the likelihood is that the number of threads able to be run will remain orders-of-magnitude higher than that available on a multi-core CPU.
- When running a custom program, the GPU can also drive the user's desktop windows. However for debugging, the GPU must be dedicated to the debugging process and cannot run the user's desktop environment. This means that for debugging (a regular programming task), either a separate, dedicated GPU must be used in the user's desktop computer or a server is used where the desktop environment is not required during debugging.

The key disadvantage of GPU computing is that programming in C is more complex than programming in IDL as the former's requirements are more rigorous. An ideal programming solution would entail a dynamically-typed, interpreted language with multi-threaded support for running functions ('kernels') on GPU hardware. This would allow scientists a flexible and high-performance means to analyze large datasets. However a 'simple to use' language with the ability to take full advantage of GPU capability is not yet present.

Despite all the above concerns, and especially now that GPUs are becoming ubiquitous in the CPU die of new processors, the time-saving benefits of GPU performance are significant. In the case of this experiment, a 2-3 hour task can be undertaken a few times a day, whereas a task of 7+ hours tends to be relegated to an overnight or all-day run. The impact means scientific dataset

analysis can occur at a pace dictated by the science rather than at pace limited by computational ability. For the current and future direction of Coral Reef Watch and other NESDIS programs, processing higher-resolution datasets and increasingly complex algorithms necessitate consideration of GPU technology to gain substantial performance benefits at a reasonable cost.

7. Acknowledgements

The authors would like to thank Gang Liu, Ph.D., and Mark Eakin, Ph.D., for providing extensive review of this document in its draft form.

8. Bibliography

Berkelmans, R. (2002) Time-integrated thermal bleaching thresholds of reefs and their variation on the Great Barrier Reef. *Mar Ecol Prog Ser* 229, 73-82.

Casey, K.S., Brandon, T.B., Cornillon, P. & Evans, R. (2010) The past, present, and future of the AVHRR Pathfinder SST Program. *Oceanography from Space, Again: Revisited* (ed. by V. Barale, J. F. R. Glover and L. Alberotanza). Springer, Verlag, Berlin.

Dennis, G.D. and R.I. Wicklund (1993) The relationship between environmental factors and coral bleaching at Lee Stocking Island, Bahamas in 1990. *In: Case Histories for the Colloquium and Forum on Global Aspects of Coral Reefs: Health, Hazards and History, 1993, F15-F21*

Hoegh-Guldberg, O. (1999) Climate change, coral bleaching and the future of the world's coral reefs. *Mar Freshwater Res*, 50, 839-866.

Kilpatrick, K. A., G. P. Podestá, and R. Evans (2001) Overview of the NOAA/NASA advanced very high resolution radiometer Pathfinder algorithm for sea surface temperature and associated matchup database. *Journal of Geophysical Research*, 106, 9179–9197.

Maynard, J.A., P.J. Turner, K.R.N. Anthony, A.H. Baird, R. Berkelmans, C.M. Eakin, J. Johnson, P.A. Marshall, G.R. Packer, A. Rea and B.L. Willis (2008) *ReefTemp*: an interactive monitoring system for coral bleaching using high-resolution SST and improved stress predictors. Geophysical Research Letters 35: L05603. doi:101029/2007GL032175.

Owens, John D., David Leubke, Naga Govindaraju, Mark Harris, Jens Kruger, Aaron E. Lefohn and Tim Purcell (2007). A Survey of General-Purpose Computation on Graphics Hardware. *Computer Graphics Forum, 26(1):80-113, March 2007.*

Skirving, W.J., A.E. Strong, G. Liu, C. Liu, F. Arzayus, J. Sapper and E. Bayler, (2006) Extreme events and perturbations of coastal ecosystems: Sea surface temperature change and coral bleaching. *Chapter 2 in Remote Sensing of Aquatic Coastal Ecosystem Processes, L.L. Richardson and E.F. LeDrew (Co-Eds), Kluwer publishers.*

Winter A., R.S. Appeldoorn, A. Bruckner, E.H. Williams and C. Goenaga (1998) Sea surface temperatures and coral reef bleaching off La Parguera, Puerto Rico (northeastern Caribbean Sea). *Coral Reefs* 17, 377-382.

NESDIS 119 **Calibration and Validation of NOAA 18 Instruments. Fuzhong Weng and Tsan Mo, December 2005.**

NESDIS 120 **The NOAA/NESDIS/ORA Windsat Calibration/Validation Collocation Database. Laurence Connor, February 2006.**

NESDIS 121 **Calibration of the Advanced Microwave Sounding Unit-A Radiometer for METOP-A. Tsan Mo, August 2006.**

NESDIS 122 **JCSDA Community Radiative Transfer Model (CRTM). Yong Han, Paul van Delst, Quanhua Liu, Fuzhong Weng, Banghua Yan, Russ Treadon, and John Derber, December 2005.**

NESDIS 123 **Comparing Two Sets of Noisy Measurements. Lawrence E. Flynn, April 2007.**

NESDIS 124 **Calibration of the Advanced Microwave Sounding Unit-A for NOAA-N'. Tsan Mo, September 2007.**

NESDIS 125 **The GOES-13 Science Test: Imager and Sounder Radiance and Product Validations. Donald W. Hillger, Timothy J. Schmit, September 2007**

NESDIS 126 **A QA/QC Manual of the Cooperative Summary of the Day Processing System. William E. Angel, January 2008.**

NESDIS 127 **The Easter Freeze of April 2007: A Climatological Perspective and Assessment of Impacts and Services. Ray Wolf, Jay Lawrimore, April 2008.**

NESDIS 128 **Influence of the ozone and water vapor on the GOES Aerosol and Smoke Product (GASP) retrieval. Hai Zhang, Raymond Hoff, Kevin McCann, Pubu Ciren, Shobha Kondragunta, and Ana Prados, May 2008.**

NESDIS 129 **Calibration and Validation of NOAA-19 Instruments. Tsan Mo and Fuzhong Weng, editors, July 2009.**

NESDIS 130 **Calibration of the Advanced Microwave Sounding Unit-A Radiometer for METOP-B. Tsan Mo, August 2010**

NESDIS 131 **The GOES-14 Science Test: Imager and Sounder Radiance and Product Validations. Donald W. Hillger and Timothy J. Schmit, August 2010.**

NESDIS 132 **Assessing Errors in Altimetric and Other Bathymetry Grids. Karen M. Marks and Walter H.F. Smith, October 2010.**

NESDIS 133 **The NOAA/NESDIS Near Real Time CrIS Channel Selection for Data Assimilation and Retrieval Purposes. Antonia Gambacorta and Chris Barnet, August 2011.**

NESDIS 134 **Report from the Workshop on Continuity of Earth Radiation Budget (CERB) Observations: Post-CERES Requirements. John J. Bates and Xuepeng Zhao, May 2011.**

NESDIS 135 **Averaging along-track altimeter data between crossover points onto the midpoint gird: Analytic formulas to describe the resolution and aliasing of the filtered results. Chang-Kou Tai, August 2011.**

NESDIS 136 **Separating the Standing and Net Traveling Spectral Components in the Zonal-Wavenumber and Frequency Spectra to Better Describe Propagating Features in Satellite Altimetry. Chang-Kou Tai, August 2011.**

NESDIS 137 **Water Vapor Eye Temperature vs. Tropical Cyclone Intensity. Roger B. Weldon, August 2011.**

NESDIS 138 **Changes in Tropical Cyclone Behavior Related to Changes in the Upper Air Environment. Roger B. Weldon, April 2011.**

NOAA SCIENTIFIC AND TECHNICAL PUBLICATIONS

The National Oceanic and Atmospheric Administration **was established as part of the Department of Commerce on October 3, 1970. The mission responsibilities of NOAA are to assess the socioeconomic impact of natural and technological changes in the environment and to monitor and predict the state of the solid Earth, the oceans and their living resources, the atmosphere, and the space environment of the Earth.**

The major components of NOAA regularly produce various types of scientific and technical information in the following types of publications

PROFESSIONAL PAPERS – Important definitive research results, major techniques, and special investigations.

CONTRACT AND GRANT REPORTS – Reports prepared by contractors or grantees under NOAA sponsorship.

ATLAS – Presentation of analyzed data generally in the form of maps showing distribution of rainfall, chemical and physical conditions of oceans and atmosphere, distribution of fishes and marine mammals, ionospheric conditions, etc.

TECHNICAL SERVICE PUBLICATIONS – Reports containing data, observations, instructions, etc. A partial listing includes data serials; prediction and outlook periodicals; technical manuals, training papers, planning reports, and information serials; and miscellaneous technical publications.

TECHNICAL REPORTS – Journal quality with extensive details, mathematical developments, or data listings.

TECHNICAL MEMORANDUMS – Reports of preliminary, partial, or negative research or technology results, interim instructions, and the like.

U.S. DEPARTMENT OF COMMERCE
National Oceanic and Atmospheric Administration
National Environmental Satellite, Data, and Information Service
Washington, D.C. 20233